Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2012) 2135–2143
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Parallelizing discrete dislocation dynamics simulations on multi-core systems

Florina M. Ciorba, Sebastien Groh and Mark F. Horstemeyer

Center for Advanced Vehicular Systems, Mississippi State University, Mississippi State, MS 39759,{florina,groh,mfhorst}@cavs.msstate.edu, http://www.cavs.msstate.edu/

Abstract

Materials science simulations are among the leading applications for scientific supercomputing. Discrete dislocation dynamics (DDD) is a numerical tool used to model the plastic behavior of crystalline materials using the elastic theory of dislocations. DDD simulations require very long running times to produce meaningful scientific results. This paper presents early experiences and results on improving the running time of Micromegas, an application code for three-dimensional DDD simulations. We used open source profiling and tracing tools to analyze the behavior and performance, as well as to identify the performance bottlenecks of Micromegas. The major performance bottleneck of Micromegas, amounts to ~68% of the total sequential run time and is parallelized using OpenMP. Evaluation and validation tests conducted on a Nehalem quad-core processor show ~50% improvement in the simulation time for 3-D DDD over 100,000 time steps. The correctness of the scientific data produced by the parallel Micromegas are successfully validated against those of the serial version.

Keywords: discrete dislocations, Micromegas, TAU, Jumpshot, OpenMP, parallelization, Intel Nehalem

1. Introduction

Computational simulations are a valuable approach in enabling scientists to examine the scientific phenomenon by providing more information than is available from experimental testing. Research or production scientific simulation codes are usually 'legacy' codes developed over many years by multi-member teams. Advances in computing systems, however, occur at a much faster rate. Therefore, there is an ever widening gap between 'legacy' scientific codes and the computing systems they are executed on. This gap can be closed via a 'performance lift' approach comprising various performance optimizations (compiler-based optimization, multi-threaded parallelizations, etc.). In this paper we 'lift' the computational performance of Micromegas to the computing levels of multi-core systems.

1.1. Description of the problem

The application code is at the heart of any predictive computational simulations. Micromegas is a legacy application code used to study the plasticity of monocrystalline metals. In crystalline materials, plastic deformation may be explained by (i) twinning, (ii) martensic transformation or/and (iii) dislocation interactions (see Figure 1). Micromegas is a code based on the elasticity theory that models the dislocation interactions into an elastic continuum. The complex nature of scientific phenomena captured by Micromegas, in conjunction with its serial implementation yield very long running simulations, which, depending on the input parameters, can take *up to a month* until they

1877-0509
doi:10.1016/j.procs.2010.04.239

produce the desired information. The desired information is usually in the form of a higher strain such that stage II or III of the stress-strain curve is reached.

The computational overhead in Micromegas is due to the long-range character of the dislocation stress field. This overhead is distributed between part (1) *calculating the interaction force between the dislocation segments* using the elasticity theory and part (2) *handling the reactions between dislocations* generated by the core properties and implemented through a set of local rules. In part (1), called FORCE, the driving force on the dislocation segment is the combination of the following: the interaction force between dislocations, the self-force and the projection on the slip system of the applied force. The interaction force between dislocations is calculated using the formula in [3]. The computation of the interaction force between dislocations represents the most expensive calculations in the simulation. The computational complexity of calculating the interaction force is $O(N^2)$, where the number of dislocation segments, N, increases during the simulation (see Figure 7). Contrary to the molecular dynamics methodology where the number of atoms is constant, here the number of segments increases with plastic deformation. This renders Micromegas a very interesting application for the computer science community. The computational efficiency of calculating the dislocation elastic field can be improved using a multipolar expansion method [8]. Using this method, the complexity is reduced from $O(N^2)$ to O(N), with an error of 0.1%. Even in this case, however, the simulation is still limited to less than 0.5% of the plastic deformation [10].

In part (2), called UPDATE, handling the reactions between dislocations requires powerful computational resources for operations of a different type than the ones used in the first part. The reason is that UPDATE is not based on analytical formulae. A specific algorithm is used to perform an exhaustive search for dislocation segments present in the area swept by the moving segment. Whenever segments are detected in that area, a series of different possible reactions involving the moving segment and the obstacle segment are tested, and the one with the lowest energy will be formed. The characteristics of the dislocation segment are then updated and the resulting plastic strain is calculated accordingly. The computational complexity of this part of the code is also $O(N^2)$.

1.2. Motivation and contribution

The performance of Micromegas is influenced by the parameters of the problem, which, unfortunately, evolve during the course of the simulation. The evolving nature of the application is due to the fact that *N* (total number of segments), and *ND* (the number of moving segments with length greater than zero) take different values after every simulation time step, following an increasing trend as illustrated in Figure 7 (explained later in the paper). Therefore, the problem size and required computational resources become known only at the beginning of a new time step. The number of time steps in a typical simulation run ranges from 10^4 to 10^9 steps. Additionally, small source code changes can lead to significant performance changes, including performance degradation. Thus, *analyzing and improving the performance of 3-D DDD simulations across a variety of existing computing systems constitutes a topic of research*. New algorithms and methods are needed to efficiently simulate the longtime behavior of dislocations, an area that has proven to be less amenable to parallelization than large system size problems. This work is the first to attempt the parallelization of Micromegas, in general, and on multi-core architectures, in particular.

The behavior and performance of Micromegas is analyzed using TAU [12], an open source performance measurement system. The use of TAU allowed for identification of the major performance bottlenecks. The main contribution of the paper is parallelizing the most important bottleneck (FORCE) using a multi-threaded fork and join approach. The parallelization significantly reduced the overhead of FORCE, while the total simulation time was reduced by ∼50% on 4 threads. The scientific simulation data (e.g., stress, strain and dislocation density) of the parallel simulation are successfully validated for correctness against those of the sequential simulation.

The rest of this paper first discusses related work in Section 2, and then describes our parallelization and implementation in Section 3. Section 4 describes how we evaluated our parallelization and implementation and presents the results obtained. Section 5 presents our conclusions and describes future work.

2. Related Work

It has been shown that the numerical limitations associated with 3-D discrete dislocation dynamics may be overcome by the use of parallel computing. Rhee et al. [6] developed a parallel version of the DDD code called '*micro3d*'. The parallel code yielded a significant speedup, which did not scale with the number of processors. The saturation of

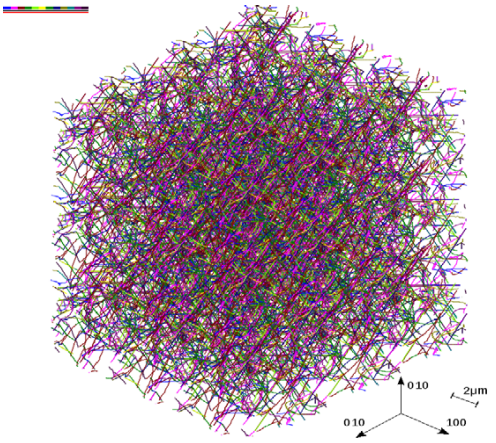


Figure 1: 3-D simulation box with a dislocation density of $10^{12}/m^2$.

```
! Module MAIN: simulation time loop
TIME: do = 1, STEPS
...
  call SOLLI  ! Apply load
  call DISCRETI ! Discretize the simulation volume
     !into dislocation lines/segments
  call FORCE ! Calculate interaction forces
     !FORCE calls SIGMA_INT_CP to calculate short
     !range interaction forces
     !FORCE calls SIGMA_INT_LP to calculate long
     !range interaction forces
  call DEPPREDIC ! Predict moving segments
  call UPDATE ! Search for obstacles, determine &
     !make contact reactions,
     !update positions of segments
  call CORRIGER_CONFIG ! Check the connections
     !between all segments
   ...
enddo TIME
```

Figure 2: Serial Micromegas: Pseudocode of the MAIN Micromegas module illustrating the most important subroutines called during each simulation time step.

parallelism is mainly due to the short-range reactions between the dislocation segments, which necessitates *frequent* communication among processors. Nevertheless, using 'micro3d', large-scale dislocations problems and dislocations-defects problems were analyzed [5] for materials with FCC (face-centered cubic) crystal structure. Recently, Shin et al. [7] proposed a parallel algorithm to speed up the edge-screw model (dynamics of segments), also for materials with FCC crystal structure. Wang et al. [9] and Arsenlis et al. [2], however, proposed a parallel algorithm to speed up the cubic and linear splines models (dynamics of nodes), for materials with BCC (body-centered cubic) crystal structure. Using the parallel version of the DD simulation code, Arsenlis et al. [2] were able to reach 1.7% of the plastic deformation during the tensile test of a specimen of molybdenum at an elevated temperature. Senger et al. [11] used a parallel DDD code to study stress distributions in polycrystalline materials with FCC crystal structure. The interested reader is referred to a recent review by Groh and Zbib [4] of the discrete dislocation methodology and its implication to multiscale modeling of the mechanical behavior of crystalline materials.

Until today, the development of Micromegas was focused on the physics involved in the plastic deformation of various crystal structures (FCC, BCC, HCP), while all the above parallel codes are dedicated to a single crystal structure.

3. Parallelization of Micromegas

Micromegas is written in a mix of Fortran 90 and Fortran 95, consists of 16 source modules and contains roughly 25,000 lines of code. Figure 2 gives the pseudocode of the `MAIN` module in Micromegas. A typical simulation run in Micromegas requires somewhere between 10^6 to 10^9 time steps to gain more insight about the plastic deformation range. Simulations with a smaller number of steps will very likely not capture the plastic range of deformation – the region of interest for the materials scientists studying plastic deformation. The simulations presented in this paper are over 10^5 time steps, each time step corresponding to 10^{-9} seconds. A simulation run over 10,000 steps using serial version of Micromegas requires *68 hours* on average and reaches 0.2% of the plastic deformation (strain %, see Figure 8) on a Nehalem quad-core Xeon W3570 processor, with 6GB of triple channel 133MHz DDR-3 RAM. As explained in Section 1.1 and Section 1.2, simulations of about 10^9 time steps are needed to reach the desired percentage of deformation, that is, the strain rate as high over 1% as possible.

3.1. Analyzing the serial application

To achieve our goal, we begin by extensive measures of Micromegas' performance using TAU [12], a state-of-the-art portable profiling and tracing package. TAU allowed easy and customizable instrumentation of Micromegas. We ran Micromegas instrumented with TAU over 100,000 time steps. The instrumented version of the original code

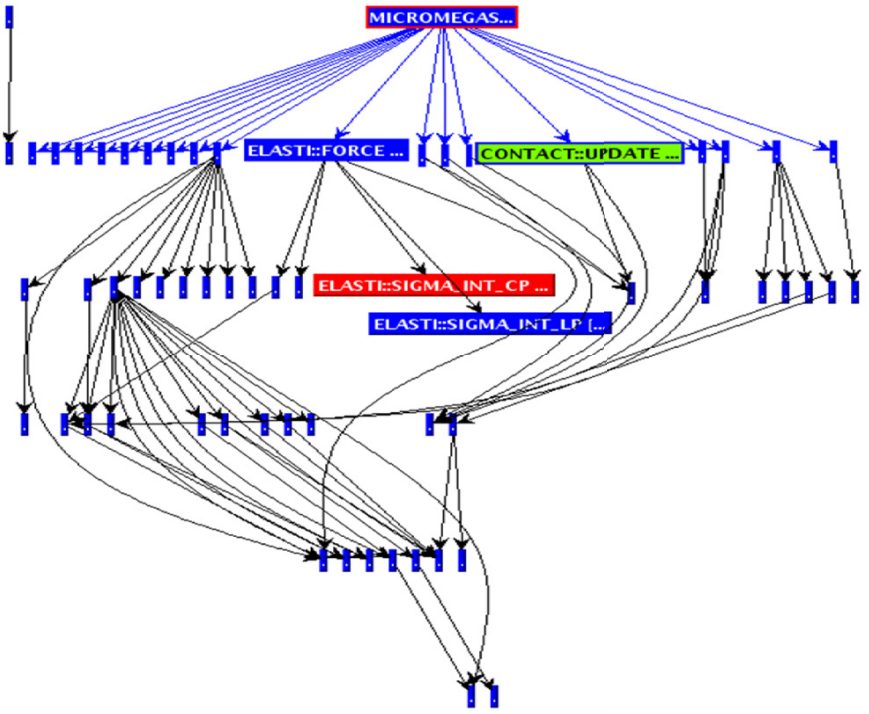


Figure 3: Serial Micromegas: Visualization of serial execution callgraph for 100,000 time steps. Important subroutines are highlighted in colors, where red denotes a 'hot spot', green a 'warm spot' and blue a 'cold spot'.

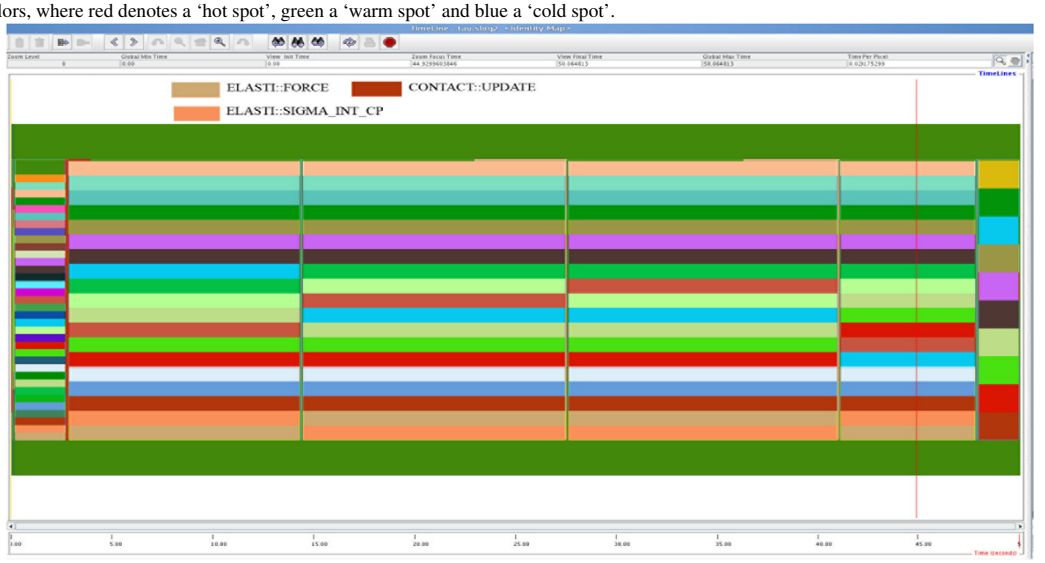


Figure 4: Serial Micromegas: Trace visualization of serial execution for 10 time steps. The legend highlights FORCE in baige, SIGMA_INT_CP in orange and UPDATE in dark red color. SIGMA_INT_CP and UPDATE are executed for most of the execution on a single CPU.

was executed on a single core of the machine. TAU enabled us to collect profiling information at various levels: outer loops, routine level and memory leaks. Using TAU, we were also able to collect traces of the serial execution of the instrumented code. Being able to obtain these measurements was crucial to understanding the behavior of the simulation code and to identifying the performance bottlenecks of the application.

We collected profiling information from a serial run of the application over 100,000 time steps. Profiling yields timing information summed over all invocations of a function. The execution callgraph of the original application code is illustrated in Figure 2. The callgraph was illustrated with Paraprof, TAU's 3-D profiling data visualization tool. The red box indicates the most time consuming part of the application, i.e., the 'hot spot', namely subroutine SIGMA_INT_CP of module ELASTI. The green box indicates the second most time consuming part of the application, i.e., the 'warm spot', namely subroutine UPDATE in module CONTACT. Blue color indicates subroutines that take less time than those in the green or red color boxes, and are called 'cold spots'. Based on the profiling information obtained with TAU, we were able to sort the modules of Micromegas in decreasing order of importance, with respect to the percentage of the total serial time attributed to each module. The profiling data for the simulations over 100,000 time steps indicated that the most important module was ELASTI, and the most important subroutine of this module was SIGMA_INT_CP, called by subroutine FORCE. Similarly, the second most important module was CONTACT with the most important subroutine of this module being UPDATE. Subroutine ELASTI::SIGMA_INT_CP accounts for 72.675% of the serial run time, while CONTACT::UPDATE is responsible for 26.885% of the serial run time, respectively.

Tracing yields a time line of events occurring throughout the execution of an application. The traces produced by TAU can be converted to various formats, that can be interpreted by other trace visualization tools. Due to the fact that tracing is a lower level type of application performance analysis, the volume of trace data is much higher than that of profiling data. Using TAU, we collected traces of the serial execution of the application only for 10 time steps. These traces were converted to the SLOG2 format in order to be visualized with Jumpshot [13]. Figure 3 illustrates the serial execution trace visualized with Jumpshot. The legends highlight the most time consuming subroutines, i.e., FORCE, SIGMA_INT_CP and UPDATE.

Upon analyzing both the profiling and tracing data, and based on Figure 2, we were able to identify the fact that to speed up the calculations of a single time step, our efforts need to be focused on speeding up SIGMA_INT_CP, which calculates the short range interaction forces between the dislocation segments. Senger et al. [11] used a similar idea to parallelize the interaction forces calculations in their DDD code, which based on the dynamics of nodes, whereas Micromegas is based on the dynamics of segments.

3.2. Analyzing the parallelized application

Subroutine SIGMA_INT_CP contains a mix of nested DO loops and a series of consecutive DO loops that iterate over *the number of boxes*, *the number of segments* in each 3-D box and over *each segment* in a box. These nested DO loops calculate the short-range interaction force between the segments, and, therefore, the iterations of these loops are independent of each other. We parallelized the outermost loop, which iterates over the number of boxes of the simulation domain, by distributing its iterations among the processing cores in a load balanced fashion.

Following parallelization, the next important step was to ensure the thread safety of the parallelized application code. This requires a good knowledge of the application code and of its behavior. Failing to ensure the thread safety of the parallelized code has a direct impact on both the performance and the correctness of the parallelized application code.

After parallelization and thread safety, we collected profiling data for the execution of the parallelized Micromegas. Figure 5 shows the execution callgraphs of the master thread (left) and worker threads (right). In the fork-and-join model, only the master thread 'out-lives' the execution of a parallelized code region, while the worker threads 'are alive' only inside a parallelized code region. The callgraph of the master thread subsumes the worker thread callgraph. This is due to the fact that inside a parallel code region the master thread behaves just like a worker thread, as dictated by the fork-and-join model. This explains the close similarity between the callgraph of the master thread and the callgraph of the main thread of the serial execution from Figure 3. The callgraph of a worker thread is the same for all worker threads.

Using TAU, we collected traces of the parallel execution for 10 time steps. Figure 4 illustrates the parallel execution trace visualized with Jumpshot. The parallel trace shows the execution of the four threads, where three of the

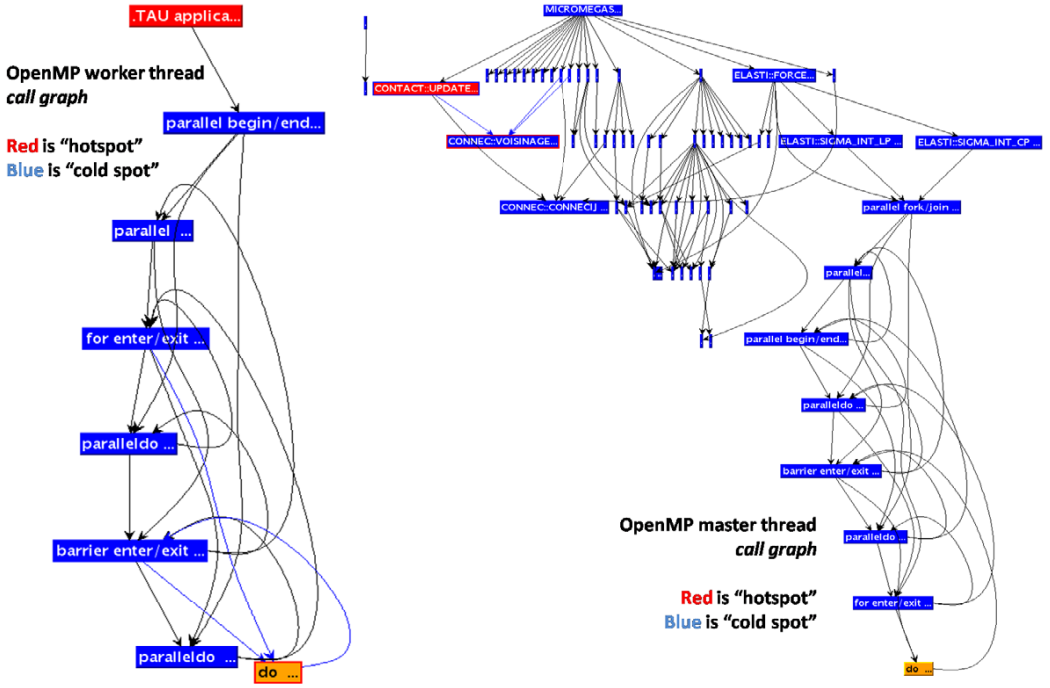


Figure 5: Parallel Micromegas: Callgraphs visualization for parallel execution on 4 cores - master thread callgraph (*right*) and the worker threads callgraph (*left*) for 100,000 time steps.

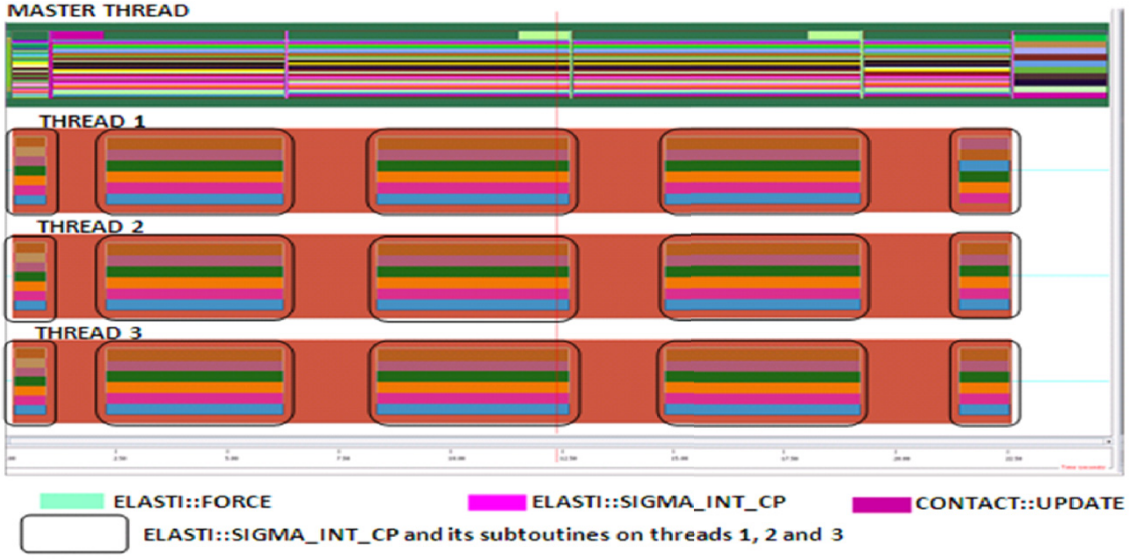


Figure 6: Parallel Micromegas: Trace visualization of parallel execution for 10 time steps on 4 CPUs. The legend highlights FORCE, SIGMA_INT_CP and UPDATE. SIGMA_INT_CP is executed in parallel.

threads are created to execute SIGMA_INT_CP in parallel with the master thread. The master thread is responsible for the serial execution of all other routines.

Analogous to Figure 3, the red box in Figure 5 (right) indicates the most time consuming subroutine of the application. In this case, however, the red box indicates the subroutine UPDATE, which in the serial code was indicated in a green box. Also, subroutine SIGMA_INT_CP is no longer indicated in a red box but in a blue box, which signifies that it is no longer the most time consuming part of the application.

Regard the work distribution methods, we chose static scheduling for distributing the iterations of the outermost loop in SIGMA_INT_CP, which iterates over the number of boxes. The chunk size was equal to the number of boxes divided by the number of threads. Then, we used selected another chunk size equal to half the original chunk size, that is, the number of boxes divides by twice the number of threads. The results presented in this paper were obtained with static scheduling and the second choice of chunk size. Due to the fact that all cores are homogeneous and due to the dedicated test system, the master thread executes SIGMA_INT_CP in a perfectly load balanced fashion simultaneously with the worker threads.

3.3. Verifying the correctness of the parallelized application code

To verify the correctness of the parallel application code we conducted two sets of serial and parallel tests. The goal of the first set was to verify the total number of segments, N, and the number of moving segments with length greater than zero, ND, produced by the serial and parallel application codes. The tests indeed verify that evolution of N and ND over 100,000 time steps is the same for both serial and parallel application and is depicted in Figure 7. The second set, shown in Figure 9, verifies that the mechanical properties produced by the serial and the parallel version of Micromegas are the same.

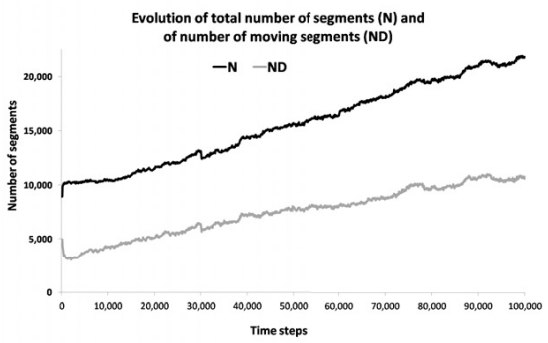
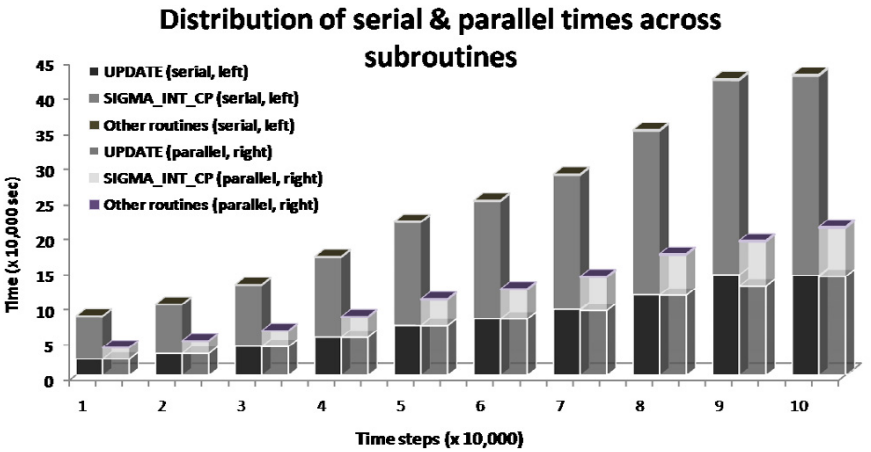


Figure 7: Evolution of the total number of segments, N, and the number of moving segments, ND, over 100,000 time steps. N and ND from the parallel simulation run are equal to those from the serial simulation run.

4. Evaluation

Parallelizing Micromegas required analyzing the original application code and evaluating its behavior in conditions similar to those of a representative DDD simulation. The simulation parameters of a representative Micromegas simulation are: 0.5% of plastic deformation, in a box of dimension 10x10x10 μm^3 with an initial density of 10^{12} m^{-2} and a strain rate of 10 s^{-1} in multislip conditions. Multislip calculations were performed to evaluate and demonstrate the efficiency of the parallel version of Micromegas. Representative volume elements of Al (FCC crystal structure with Burgers vector of magnitude b = 2.86 Å) of dimensions 9x10x12 μm^3 were loaded along the [001] direction, with a strain rate of 20 s^{-1} at a temperature of 300 K under periodic boundary conditions. Screw dislocations were not allowed to cross-slip at any time, and the time step was considered to be 10^{-9} seconds.

The evaluation tests were conducted on a dedicated commodity Nehalem quad-core Xeon W3570 processor, running at 3.2GHz with 6GB DDR-3 RAM, SLES 10 OS, 2.6.16.60 Linux kernel. The speedup and efficiency of the parallel application code relative to the serial application code are of interest for the evaluation tests.



Time steps	10,000	20,000	30,000	40,000	50,000	60,000	70,000	80,000	90,000	100,000
Speedup	2.14	2.07	2.03	2.02	2.01	2.03	2.04	2.02	2.20	2.02
Efficiency	0.53	0.52	0.51	0.51	0.50	0.51	0.51	0.51	0.55	0.51

Figure 8: Distribution of the serial and parallel execution times across code subroutines. For every simulation run, the time spent in SIGMA_INT_CP is significantly decreased when running on 4 CPUs, which yields a speedup of a factor of 2 for the corresponding parallel time of a simulation run. The total time over 100,000 time steps is, therefore, decreased from *28 days* (serial) to *13.8 days* (parallel).

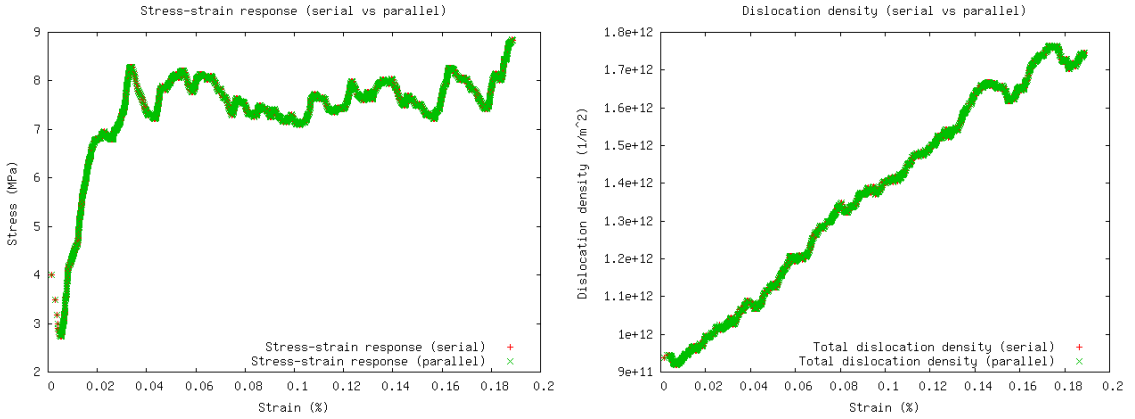


Figure 9: Validation of scientific data obtained from the parallel simulation against the serial simulation over 100,000 time steps. *Left*: Stress vs. strain. *Right*: Evolution of the total dislocation density as a function of the plastic strain.

During profiling, it was determined that on average SIGMA_INT_CP, accounts for ~67.40% of the serial execution time. Attributing the parallelizable part of Micromegas to SIGMA_INT_CP, one can say that S=32.60% is the serial part of Micromegas. Assuming P=4 available processing elements, the speedup through parallelism on these processing elements, as dictated by Amdahl's law [1], is $1/(S + ((1 - S)/P) = 2.02$, while the theoretical speedup limit assuming infinite parallelism, $P- > \infty$, is then $1/S = 3.06$.

Figure 8 shows the distribution of the serial and parallel execution times across the code subroutines. The speedup and efficiency demonstrate the scalability of the parallel application code relative to the serial application code when

the number of time steps increases from 10,000 to 100,000 time steps.

5. Conclusions and Future Work

Advancements in application analysis tools have made parallelization of complex application codes an easier task than before. Good implementation practices together with a fair knowledge of the application code, however, are imperative for making parallelization a less complicated and faster task.

In this paper we narrowed the gap between Micromegas and multi-core computing systems. We achieved this via a 'performance lift' based on the multi-threaded parallelization of the major performance bottleneck of the original application code. The outcome of this work is an improved simulation code for research in discrete dislocation dynamics for materials science. The significance of this work lies in that the improvements in the simulation code permit the scientist to perform longer simulations of higher temporal fidelity, using more processors, which were not feasible previously.

The results presented in this paper are the first efforts to parallelize Micromegas. There is certainly room for improving the current results that could be addressed via several directions, which may include parallelization of the UPDATE subroutine, different scheduling methods for the parallelized subroutines, scaling the parallel code beyond 4 cores, or even using graphics-based accelerators. Such directions will be addressed in future work.

6. Acknowledgments

The authors would like to thank the developers of TAU for their valuable support in installing and configuring the program and performance analysis tool framework. The funding provided for this study by the US Department of Energy under Grant No. 008860-013 and by the Center for Advanced Vehicular Systems of the Mississippi State University is gratefully acknowledged.

References

[1] Amdahl, G., 1967. "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings, (30), pp. 483-485.
[2] Arsenlis, A., Cai, W., Tang, M., et al., 2007. "Enabling strain hardening simulations with dislocation dynamics," *Modelling Simul. Mater. Sci. Eng. 15*, pp. 553-95.
[3] Devincre, B., 1995, "Three dimensional stress fields expressions for straight dislocation segments." Solid State Communication 93, pp. 875-8.
[4] Groh, S. and Zbib, H.M., 2009. "Advances in discrete dislocations dynamics and multiscale modeling," J. Eng. Mat. Tech. 31, pp. 041209-1.
[5] Khraishi, T, Zbib, H.M., T. Diaz de la Rubia and M. Victoria, 2002, "Localized Deformation and Hardening In Irradiated Metals: Three-Dimensional Discrete Dislocation Dynamics Simulations," Metallurgical and Materials Transactions B, 33B, pp. 285-96.
[6] Rhee, M., Zbib, H.M., Hirth, J.P., Huang, H. and de La Rubia T. D., 1998. "Models for Long/Short Range Interactions in 3D Dislocation Simulation." Modelling Simul. Mater. Sci. Eng., 6, pp. 467-92.
[7] Shin, C.S., Fivel, M.C., Verdier, M. and Kwon, S.C., 2006. "Numerical methods to improve the computing efficiency of discrete dislocation dynamics simulations." J. Comp. Physics, 215, pp. 417 - 429.
[8] Verdier, M., Fivel, M. and Groma, I., 1998. "Mesoscopic scale simulation of dislocation dynamics in fcc metals: principles and applications." Modelling Simulation Mater. Sci. Eng. 6, pp. 755770.
[9] Wang Z.Q., Beyerlein I.J. and LeSar R., 2007. "The importance of cross-slip in high rate deformation." Modelling Simul. Mater. Sci. Eng. 15, pp. 675-690.
[10] Zbib, H. M., de La Rubia, T. D., Rhee, M., Hirth, J. P., "3D Dislocation Dynamics: Stress-Strain behavior and Hardening Mechanisms in FCC and BCC Metals", J. Nuc. Maters., 276, pp. 154-165, 2000.
[11] Senger, J., Augustin, W., Weygand, D. M., Kraft, O., Heuveline, V., Gumbsch, P., 2006. "Parallel Discrete Dislocation Dynamics: Stress Distribution in Polycrystalline Metallic Film." Multiscale Modeling of Materials, Materials Research Society Fall 2006 Meeting.
[12] TAU Performance System, http://www.cs.uoregon.edu/research/tau/home.php
[13] Performance Visualization for Parallel Programs, online, [http://www.mcs.anl.gov/research/projects/perfvis/software /viewers/index.htm#Jumpshot]